|  |  |
| --- | --- |
| Изображение выглядит как герб, эмблема, символ, нашивка  Автоматически созданное описание | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ «ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ»

# КАФЕДРА «ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ»

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2 ПО ДИСЦИПЛИНЕ:**

**ТИПЫ И СТРУКТУРЫ ДАННЫХ**

***<Записи с вариантами. Обработка таблиц*>**

Студент **<*Ермаков И.Г*>**

Группа **<*ИУ7 -32Б*>**

Название предприятия **НУК ИУ МГТУ им. Н. Э. Баумана**

|  |  |
| --- | --- |
| Студент | **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ <Ермаков И.Г>** |
| Преподаватель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **<Фамилия ИО>** |

**2024**

**Содержание**

Условие задачи…………………………………………………………………….....3

Описание ТЗ……………………………………………………………………….....3

Входные данные……………………………………………………………….3

Выходные данные……………………………………………………………..3

Обращение к программе……………………………………………………...4

Аварийные ситуации………………………………………………………….4

Описание структуры данных………………………………………………………..6

Время сортировки таблиц…………………………………………………………...8

Тесты………………………………………………………………………………...12

Выводы……………………………………………………………………………...13

Ответы на контрольные вопросы………………………………………………….14

**Условие задачи**

Создать таблицу, содержащую не менее 40-ка записей (тип – запись с вариантами

(объединениями)). Упорядочить данные в ней по возрастанию ключей, двумя алгоритмами

сортировки, где ключ – любое невариантное поле (по выбору программиста), используя:

а)саму таблицу

б) массив ключей.

Возможность добавления и удаления записей в ручном режиме, просмотр таблицы, просмотр

таблицы в порядке расположения таблицы ключей обязательна.

Вывод списка квартир, содержащий: адрес, общую площадь, количество комнат, стоимость квадратного метра, тип:

1. Первичное

a. С отделкой / без отделки

2. Вторичное

a. Год постройки

b. Количество предыдущих собственников

c. Были ли животные

Вывод вторичное двухкомнатное жилье в указанном ценовом диапазоне без животных

**Описание ТЗ**

**Входные данные:**

Пользователь вводит число(опцию) от 1-10. Программа в зависимости от выбранной опции решает ту или иную задачу. К каждому действию пользователю предоставляется подробное приглашение к выводу.

**Выходные данные:**

Для каждой опции пользователю выводится информация об успешности или ошибке работы программы. В случае успешности программа в зависимости от опции выводит пользователю на экран.

**Действие программы:**

Программа работает до тех пор пока пользователь не введет 0. В случае некорректного ввода опции программа предлагает набор доступных опций и приглашение к вводу для повторного ввода опции. Так же при ошибке ввода в процессе работы с таблицей (например неправильный формат года) программа вернется в исходное состояние выбора опций, описанное ранее.

**Обращение к программе:**

Программа запускается из терминала по команде ./app.exe

**Аварийные ситуации:**

ERROR\_READING\_TYPE\_ROOM (1): Ошибка чтения типа жилья. Происходит при некорректном или неуспешном вводе типа жилья (например, "primary" или "secondary").

ERROR\_READING\_ADDRESS (2): Ошибка чтения адреса. Возникает, если при вводе или чтении данных адреса произошла ошибка.

ERROR\_READING\_SQUARE (3): Ошибка чтения площади. Происходит при некорректном вводе значения площади.

ERROR\_READING\_AMOUNT\_ROOMS (4): Ошибка чтения количества комнат. Возникает, если данные о количестве комнат некорректны или не могут быть прочитаны.

ERROR\_READING\_COST (5): Ошибка чтения стоимости за квадратный метр. Происходит, если введены неверные или недопустимые данные.

ERROR\_READING\_TRIM (6): Ошибка чтения состояния отделки. Возникает, если данные об отделке некорректны.

ERROR\_READING\_YEAR (7): Ошибка чтения года постройки. Происходит при некорректном вводе года или при его отсутствии.

ERROR\_READING\_LAST\_OWNERS (8): Ошибка чтения количества предыдущих владельцев. Возникает, если введённые данные о количестве владельцев неверны.

ERROR\_READING\_PETS (9): Ошибка чтения данных о наличии домашних животных. Происходит, если данные о домашних животных некорректны.

ERROR\_VALUE\_PETS (10): Некорректное значение для данных о наличии домашних животных. Возникает, если введённое значение не соответствует допустимому диапазону (например, ввод другого символа вместо "1" или "0").

ERROR\_MATCHING\_REG\_EXPR (11): Ошибка сопоставления с регулярным выражением. Происходит, если введённые данные не соответствуют ожидаемому формату, проверяемому с помощью регулярных выражений.

ERROR\_READING\_FILENAME (12): Ошибка чтения имени файла. Возникает, если имя файла не может быть прочитано (например, файл не существует).

ERROR\_OPENING\_FILE (13): Ошибка открытия файла. Происходит, если файл не удаётся открыть (например, файл отсутствует или нет прав доступа).

ERROR\_READING\_AMOUNT\_OF\_STRUCTS (14): Ошибка чтения количества структур. Возникает, если данные о количестве структур не могут быть прочитаны.

ERROR\_INCORRECT\_AMOUNT\_SCRUCTS (15): Некорректное количество структур. Происходит, если количество введённых структур превышает или не соответствует допустимому значению.

ERROR\_EXCESS\_STRUCTS (16): Превышение допустимого количества структур. Возникает, если количество структур превышает заранее установленное ограничение.

ERROR\_READING\_STRUCT (17): Ошибка чтения структуры. Происходит, если структура данных не может быть корректно прочитана или загружена.

**Описание структур данных**

|  |
| --- |
| typedef struct keys\_t  {  int index;  int rooms\_quantity;  } keys\_t; |

|  |
| --- |
| typedef struct primary\_t  {  short trim;  } primary\_t; |

|  |
| --- |
| typedef struct secondary\_t  {  char build\_year[MAX\_LEN\_YEAR + 1];  int quantity\_prev\_owners;  short is\_pet;  } secondary\_t; |

|  |
| --- |
| typedef union choose\_type\_t  {  primary\_t prime;  secondary\_t second;  } choose\_type\_t; |

|  |
| --- |
| typedef struct desc\_t  {  char type\_room[MAX\_LEN\_TYPE\_ROOM + 1];  char address[MAX\_LEN\_ADDRESS + 1];  float area;  int rooms\_quantity;  float square\_cost;  choose\_type\_t type;  } desc\_t; |

Структура desc\_t содержит такие поля как тип жилья (type\_room), нужен для первоначального определения вложенного типа основной структуры. Поле address определяет адрес. Поле area определяет жилую площадь помещения. Поле rooms\_quantity определяет кол-во комнат в квартире. Поле square\_cost определяет стоимость квадратного метра. Поле type предназначено для следующего уровня вложенности структуры, точнее обьединение для выбора первичного или вторичного типа жилья в зависимости от ввода пользователя.

Переходим к обьединению choose\_type\_t. Это обьединение содержит два поля : prime и second соответственно отвечающие за первичный или вторичный тип жилья, который описывает следующий уровень вложенности : структуры primary\_t и secondary\_t

Структура primary\_t содержит в себе одно поле trim, которая в программа может принимать только два значения : 0 или 1. 0 – отделки нет, 1 – отделка есть

Структура secondary\_t содержит поля build\_year, quantity\_prev\_owners, is\_pet

Поле build\_year содержит информацию о годе постройки жилья

Поле quantity\_prev\_owners содержит в себе информацию о кол-ве предыдущих собственников жилья

Поле is\_pet говорит о том, жили в квартире какие то животные. 0 – нет, 1 – да

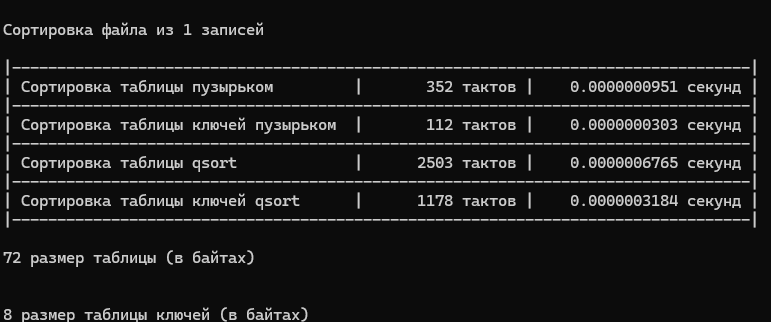
|  |
| --- |
| #define MAX\_LEN\_ADDRESS 25  #define MAX\_LEN\_YEAR 10  #define MAX\_LEN\_FILENAME 15  #define MAX\_LEN\_TYPE\_ROOM 10 |

Реализация работы программы подразумевает описанные выше константы для взаимодействия с полями структур.

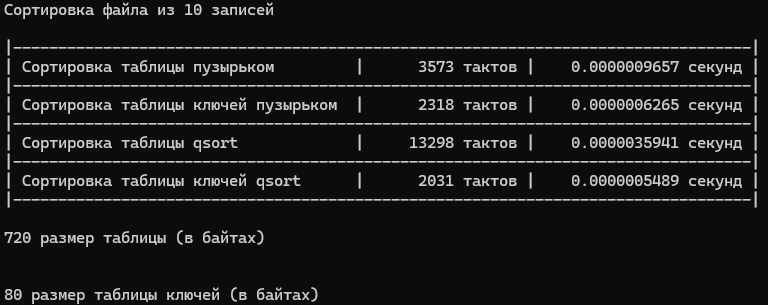
**Время сортировки таблиц**

Ниже представленны сортировки для размеров массивов (1, 10, 100, 250, 500, 1000)

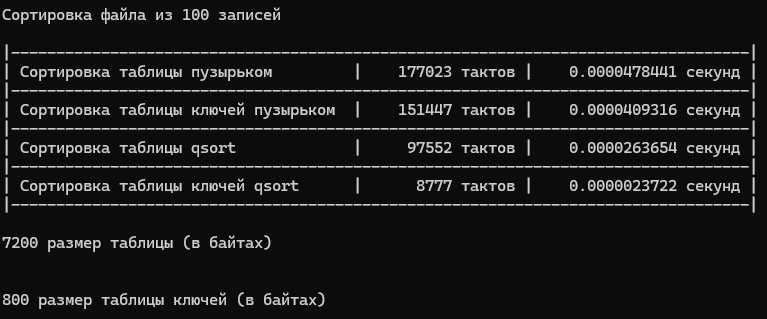
Сортировка для 1 записи



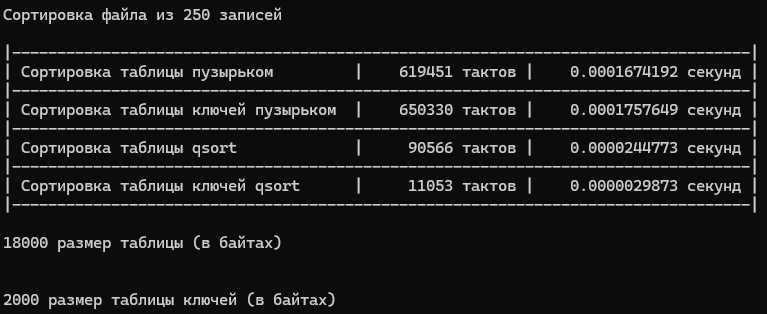
Сортировка для 10 записей



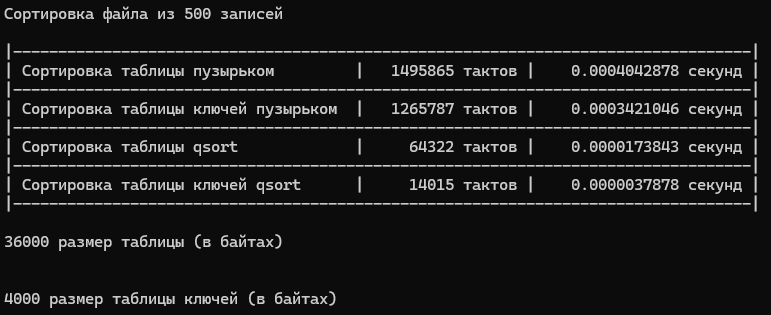
Сортировка для 100 записей



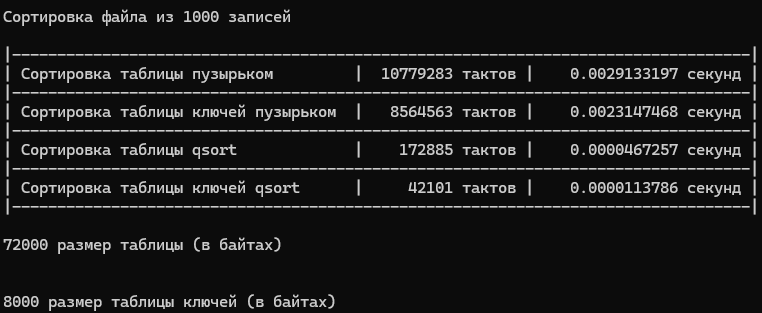
Сортировка для 250 записей



Сортировка для 500 записей



Сортировка для 1000 записей



На основании проведённых тестов можно сделать общий вывод: сортировка с использованием алгоритма быстрой сортировки (qsort) существенно превосходит пузырьковую сортировку по времени выполнения, особенно при работе с большими массивами данных. Это подтверждается результатами для различных размеров массивов.

Для массивов из 1000 записей время выполнения qsort на данных оказалось более чем в 62 раза быстрее по сравнению с пузырьковой сортировкой (10779283 тактов против 172885 тактов). Для сортировки ключей разница оказалась ещё более впечатляющей: qsort выполнялся более чем в 200 раз быстрее, чем пузырьковая сортировка ключей (8564563 тактов против 42101 тактов).

Размер таблицы ключей составляет около 11% от размера исходного массива данных, что делает сортировку по ключам более экономичной не только по времени, но и по памяти. Это особенно важно для больших объемов данных, где работа с полным массивом требует значительно больше ресурсов.

Сортировка qsort универсальна и эффективна, но требует использования динамической памяти (кучи), что может создавать дополнительные накладные расходы на её выделение и управление. Тем не менее, несмотря на эти накладные расходы, qsort остаётся лучшим выбором по сравнению с пузырьковой сортировкой, особенно при больших объемах данных.

В итоге, для любых задач, связанных с сортировкой больших объемов данных, qsort показывает наилучшие результаты по производительности, в то время как пузырьковая сортировка может быть применима лишь для небольших наборов данных, где её простота может компенсировать низкую эффективность.

Сравнение времени работы bubble\_sort и qsort для массива и ключей

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во записей | Время работы bubble\_sort для массива ключей относительно qsort для ключей | Время работы bubble\_sort для обычного массива относительно qsort для массива | Время работы qsort для массива ключей | Время работы qsort для обычного массива |
| 1 | 29% | 14% | 100% | 100% |
| 10 | 114% | 27% | 100% | 100% |
| 100 | 202% | 181% | 100% | 100% |
| 250 | 234% | 646% | 100% | 100% |
| 500 | 208% | 230% | 100% | 100% |
| 1000 | 204% | 6250% | 100% | 100% |

**Тесты**

|  |  |  |
| --- | --- | --- |
| Описание теста | Входные данные | Вывод |
| Ввод пункта меню | -1 | Ошибка ввода опции! Введите правильное значение от 0 до 10 |
| Ввод пункта меню | 11 | Ошибка ввода опции! Введите правильное значение от 0 до 10 |
| Ввод пункта меню | egsg | Ошибка ввода опции! Введите правильное значение от 0 до 10. |
| Загрузка таблицы | Кол-во жилья больше 1000 или меньше 1 | Некорретное кол-во структур в файле |
| Добавление в таблицу | Одно из полей не соответствует стандарту | Ошибка чтения параметра отделки! |
| Вывод таблицы | Если не была загружена | Пустой шаблон |
| Вывод жилья в указанном диапазоне со вторичным типом, две комнаты | Жилье не входит в диапазон, отсутствие других параметров | Не найдено ни одного элемента в таком целовом диапазоне и без животных |
| Загрузка таблицы из файла | Все указано правильно | Таблица из файла успешно считана! |
| Вывод времени сортировок таблиц | Нет данных | Ошибка, пустой файл |
| Вывод времени сортировок | Если таблица считана, данные есть | Выводится таблица с измерениями |
| Удаление записи | Ошибок нет | Запись удалена успешно! |
| Добавление записи с переполнением | Добавляется 1001 запись | Некорректное кол-во структур |

**Выводы**

Таблица, описывающая сравнение сортировок времени работы двух алгоритмов явно показывает, что с увеличением количества записей разница в производительности между qsort и пузырьковой сортировкой становится все более значительной, особенно на больших массивах данных.

Использование union в данном случае позволяет избежать выделения памяти под обе структуры сразу. Память выделяется только для одного из типов — либо для первичного жилья, либо для вторичного. Это может быть очень полезным, если в твоей программе будет обрабатываться множество записей, и разные записи могут использовать разные типы жилья.

**Ответы на контрольные вопросы**

**Как выделяется память под вариантную часть записи?**

Память под вариантную часть записи выделяется так же, как и для обычных структур, но используется общий блок памяти, который может содержать разные типы данных, в зависимости от того, какая часть активна в данный момент. Это часто делается с использованием union.

**Что будет, если в вариантную часть ввести данные, несоответствующие описанным?**

Это приведет к некорректной интерпретации данных, что может вызвать непредсказуемое поведение программы или ошибки при доступе к данным.

**Кто должен следить за правильностью выполнения операций с вариантной частью записи?**

Программист несет ответственность за корректную работу с вариантной частью записи, обеспечивая правильное использование поля, которое активно в данный момент.

**Что представляет собой таблица ключей, зачем она нужна?**

Таблица ключей — это структура данных, содержащая индексы записей и значения, по которым происходит сортировка. Она позволяет сортировать и искать данные без изменения исходной таблицы.

**В каких случаях эффективнее обрабатывать данные в самой таблице, а когда – использовать таблицу ключей?**

Эффективнее обрабатывать данные в самой таблице, если требуется частый доступ к элементам и не нужна многократная сортировка. Таблицу ключей используют, когда нужно сортировать данные несколько раз или оптимизировать операции поиска.

**Какие способы сортировки предпочтительнее для обработки таблиц и почему?**

Для больших таблиц предпочтительнее использовать алгоритмы с временной сложностью O(nlogn), такие как qsort или быстрая сортировка. Они быстрее, чем простые алгоритмы вроде пузырьковой сортировки O(n\*\*2) особенно на больших объемах данных.